

Netherlands
organization for
applied scientific
research



TNO Physics and Electronics
Laboratory



P.O. Box 96864
2509 JG The Hague
Oude Waalsdorperweg 63
The Hague, The Netherlands
Fax +31 70 328 09 61
Phone +31 70 326 42 21

TNO-report

DTIC FILE COPY

report no.
FEL-90-B023

copy no.

I.D. 90 2780

Nothing from this issue may be reproduced
and/or published by print, photoprint,
microfilm or any other means without
previous written consent from TNO.
Submitting the report for inspection to
parties directly interested is permitted.

In case this report was drafted under
instruction, the rights and obligations
of contracting parties are subject to either
the 'Standard Conditions for Research
Instructions given to TNO' or the relevant
agreement concluded between the contracting
parties on account of the research object
involved.

© TNO

8 title

BSB Radar Pulse Classification

TDCK RAPPORTCENTRALE
Frederikkazerne, Geb. 140
van den Burchlaan 31
Telefoon: 070-3166394/6395
Telefax: (31) 070-3166202
Postbus 90701
2509 LS Den Haag **TDCK**

authors:

P.P. Meiler

A.M. van Wazenberg

date : August 1990

classification

title : Unclassified

abstract : Unclassified

report : Unclassified

appendices A & B : Unclassified

no. of copies : 35

no. of pages : 44 (incl. titlepage & appendices,
excl. RDP & distribution list)

appendices : 2



All information which is classified according to Dutch regulations shall
be treated by the recipient in the same way as classified information of
corresponding value in his own country. No part of this information will
be disclosed to any party.

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited



90 09 13 211

AD-A226 472

report no. : FEL-90-B023
title : BSB Radar pulse classification

authors : P.P. Meiler, A.M. van Wezenbeek
institute : TNO Physics and Electronics Laboratory

date : August 1990
no. in pow '90 : 708

ABSTRACT (UNCLASSIFIED)

The Brain-State-in-a-Box (BSB) neural network is an auto-associative network. It is able to associate vectors with other vectors. Auto-association results in a system that has been learned to respond strongly to each input vector of a certain set of input vectors by returning this same vector multiplied by its recollection strength, where vectors are bounded by the box. This gives the possibility for convergence to one vector of the learned set of vectors if the actual input vector is not known to the system. When radar signals are translated into vectors, the BSB network can be used for radar pulse classification. The whole process of classification together with the relation with other non-neural algorithms is shown.

Signature
Franklin
1

Accession For	
NTIS	<input checked="checked" type="checkbox"/>
DTIC	<input type="checkbox"/>
Unclassified	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

rapport no. : FEL-90-B023
titel : BSB Radar pulse classification

auteurs : P.P. Meiler, A.M. van Wezenbeek
instituut : Fysisch en Elektronisch Laboratorium TNO

datum : Augustus 1990
no. in iwp '90 : 708

SAMENVATTING (ONGERUBRICEERD)

Het Brain-State-in-a-Box (BSB) neurale netwerk is een auto-associatief netwerk. Het associeert vectoren met vectoren. De auto-associatie zorgt ervoor dat een systeem wordt opgebouwd dat geleerd is om sterk te reageren op een verzameling van invoer vectoren. Dit gebeurt door bij invoer van een vector uit deze verzameling als uitvoer eenzelfde vector te genereren, vermenigvuldigd met een waarde die overeenkomt met de hevigheid van de reactie, waarbij de vector gedwongen wordt binnen de box te blijven. Dit biedt de mogelijkheid van convergentie naar één van de geleerde vectoren, ook als de actuele invoer vector niet bekend is aan het systeem. Wanneer nu radar signalen naar vectoren worden vertaald, kan het BSB netwerk gebruikt worden voor radar puls classificatie. Het gehele classificatie proces en de relatie met andere, niet-neurale, algoritmen wordt beschreven.

ABSTRACT	1
SAMENVATTING	2
CONTENTS	3
1 INTRODUCTION	4
2 DESCENT TECHNIQUES	5
2.1 Gradient descent	5
2.1.1 First order	7
2.1.2 Second order	8
2.2 Conjugate descent	8
2.2.1 Directional	9
2.2.2 Gradient	9
3 BSB NEURAL NETWORK	10
3.1 Energy curves	10
3.2 BSB implements first order gradient descent minimizing energy	11
3.3 Gradient vector orthogonal to energy field	14
3.4 Hebbian weightmatrix is positive semi-definite	14
4 BSB RADAR PULSE CLASSIFICATION	15
4.1 Translation	15
4.1.1 Thermometer coding	15
4.1.2 Reflected binary coding	19
4.2 Results of experiments	21
4.2.1 2 dimensional BSB	22
4.2.2 Clustering	24
5 CONCLUSIONS AND FURTHER RESEARCH	33
6 REFERENCES	34

APPENDIX A: LINEAR ALGEBRA

APPENDIX B: DOCUMENTATION OF C-SOURCES

1 INTRODUCTION

From the period of 1-12-1988 to 30-6-1989 the author has been involved with the study and implementation of the Brain-State-in-a-Box (BSB) neural network. This report presents both the theoretical issues and the results of practical experiments done in this period. The reader should be familiar with the basics of neural computing (see [D.E. Rumelhart, J.L. McClelland 1987] for an introduction).

Anderson showed on the conference of the INNS in Boston, 1988 that the BSB network can be used for radar signal categorisation [J.A. Anderson 1988]. This has been the direct motive for TNO to investigate its possibilities. Related articles can be found in: [J.A. Anderson 1977], [J.A. Anderson 1983], [J.A. Anderson, M.C. Mozer 1981], [J.A. Anderson, G.E. Hinton 1981].

Chapter 2 starts with a general introduction in descent techniques, of which the BSB network in fact implements the most simple one. This is shown in chapter 3. In chapter 4 it is shown how the BSB network can be used for the classification of clusters. Conclusions are given in chapter 5. Two appendices are included. Appendix A discusses those topics from linear algebra used in this report. Appendix B contains the documentation of C-sources used for the implementation of the BSB network.

2 DESCENT TECHNIQUES

This chapter contains a summary of descent techniques. All derivations can be found in [S.L.S. Jacoby, J.S. Kowalik, J.T. Pizzo 1972] or [H.W. Sorenson 1980]. Descent techniques consist of three parts. Given an initial vector x_k pointing in N-dimensional space (having N elements), a descent direction s_k (with $|s_k|=1$) must be found, which is a vector pointing towards a direction evaluating to a lower value of an objective function E. This can also be interpreted as finding a point with less energy. Second, a descent step length γ_k must be found, indicating the length of the step taken in the direction determined by the s_k vector. Finally the descent step must be made according to formula 2.1 so that $E(x_{k+1}) < E(x_k)$.

$$x_{k+1} = x_k + \gamma_k s_k \quad (2.1)$$

The vector s_k is a descent direction if:

$$\begin{aligned} \lim_{\gamma \rightarrow 0} \frac{E(x_k + \gamma s_k) - E(x_k)}{\gamma} \\ = \frac{dE(x_k + \gamma s_k)}{d\gamma} \\ = s_k^T \nabla E(x_k) < 0 \end{aligned} \quad (2.2)$$

Notice that the product of descent direction and the gradient gives the directional derivative. The condition for formula 2.2 to be negative implies that the descent direction must make an angle α with the gradient, where $\pi/2 < \alpha < 3\pi/2$, and that the descent direction is therefore opposite to the gradient.

The gradient descent techniques have in common that the first or second order derivative of the objective function is used to find its minimum. For some applications it is impossible to determine the gradient. The conjugate descent technique offers an algorithm which in principle avoids the calculation of the gradient.

2.1 Gradient descent

For the gradient descent the locally steepest direction is chosen as descent direction. The following derivation shows that the choice of locally steepest direction leads to a descent direction

given by the multiplication of the inverse of a metric matrix and the gradient vector. First define the distance d between x_1 and x_2 :

$$d(x_1, x_2) = ((x_1 - x_2)^T A (x_1 - x_2))^{1/2} \quad (2.3)$$

A is a positive definite ($d > 0$, see appendix A) $N \times N$ symmetric metric matrix. So all points x at a distance δ from x_k are on a N -dimensional ellipse (see section 3.1 for an explanation of the ellipse).

$$\delta^2 = (x - x_k)^T A (x - x_k) \quad (2.4)$$

From a point x_k we take a step $\Delta x_k = \gamma_k s_k$, so that the resulting vector is on the ellipse, and we choose this Δx_k so that the associated energy E will be minimized:

$$\min_x \{ E(x_k + \Delta x_k) \mid (\Delta x_k)^T A \Delta x_k = \delta^2 \} \quad (2.5)$$

We can approximate $E(x_k)$ with the first order Taylor series expansion about x_k :

$$E(x_k + \Delta x_k) \approx E(x_k) + (\Delta x_k)^T \nabla E(x_k) \quad (2.6)$$

To minimize $E(x_k + \Delta x_k)$, we now must minimize $(\Delta x_k)^T \nabla E(x_k)$ because $E(x_k)$ is fixed. So:

$$\min_x \{ (\Delta x_k)^T \nabla E(x_k) \mid (\Delta x_k)^T A \Delta x_k = \delta^2 \} \quad (2.7)$$

One of the possibilities to find this minimum is to use the Lagrange function:

$$L(x, u) = E(x) - \sum_{j=1}^m u_j g_j(x) \quad (2.8)$$

In formula 2.8 $u = (u_1 \ u_2 \ \dots \ u_m)^T$ is a vector of Lagrange multipliers corresponding to the constrained minimization problem:

$$\min_x \{ E(x) \mid g_j(x) \geq 0, \ j = 1, 2, \dots, m \} \quad (2.9)$$

The primal function L_p is defined as:

$$L_p(x) = \max_{u \geq 0} L(x, u) \quad (2.10)$$

and the primal problem is the problem that we want to solve:

$$\min_x L_p(x) = \min_x E(x) \quad (2.11)$$

When, as in our case, the original problem is of the form $\min\{E(x) \mid g_j(x) = 0, j = 1..m\}$, the solution can be found by solving the classical saddle-point conditions:

$$\begin{aligned}\nabla_x L(x_S, u_S) &= 0 \\ \nabla_u L(x_S, u_S) &= 0\end{aligned}\quad (2.12)$$

with $L(x_S, u) \leq L(x, u) \leq L(x, u_S)$ for all $u \geq 0$ and all x , defining the saddle-point (x_S, u_S) . So:

$$L(x, u) = (\Delta x_k)^T \nabla E(x_k) - u (\Delta x_k)^T A \Delta x_k - \delta^2 \quad (2.13)$$

$$\frac{\delta L}{\delta x} = \nabla E(x_k) - 2u A \Delta x_k = 0 \quad (2.14)$$

$$\frac{\delta L}{\delta u} = (\Delta x_k)^T A \Delta x_k - \delta^2 = 0 \quad (2.15)$$

giving: (formula 2.14 rewritten)

$$\Delta x_k = (u/2) A^{-1} \nabla E(x_k) \quad (2.16)$$

Given the condition from formula 2.2 for descent directions, the direction of s_k has been found:

$$s_k = -A^{-1} \nabla E(x_k) \quad (2.17)$$

2.1.1 First order

For the first order gradient descent for s_k , given by formula 2.17, we choose $A = I$. This results in:

$$s_k = -\nabla E(x_k) \quad (2.18)$$

The choice for γ_k is done such that $E(x_k) - E(x_{k+1})$ will be maximized. Calculating the Taylor expansion of $E(x)$ about x_{k+1} :

$$\begin{aligned}E(x_{k+1}) &= E(x_k + \gamma_k s_k) \\ &= E(x_k) + \gamma_k s_k^T \nabla E(x_k) + (\gamma_k^2 s_k^T \text{Hess}(x_k) s_k) / 2 \\ &= E(x_k) - \gamma_k \nabla E(x_k)^T \nabla E(x_k) + (\gamma_k^2 \nabla E(x_k)^T \text{Hess}(x_k) \nabla E(x_k)) / 2\end{aligned}\quad (2.19)$$

We require that:

$$\frac{dE(x_{k+1}) - E(x_k)}{dy} = 0 \quad (2.20)$$

$$\gamma = \frac{\nabla E(x_k)^T \nabla E(x_k)}{\nabla E(x_k)^T \text{Hess}(x_k) \nabla E(x_k)} \quad (2.21)$$

The first order gradient descent in fact generates directions that asymptotically converge to just two directions. This has the consequence that the rate of convergence, particularly in the vicinity of local minima, is very slow. As a result this method is generally regarded as exhibiting unsatisfactory performance, especially in the vicinity of the local minimum ¹.

2.1.2 Second order

The second order gradient descent technique uses the Hessian matrix $\text{Hess}(x_k)$ as metric matrix, so formula 2.17 gives:

$$s_k = -\text{Hess}(x_k)^{-1} \nabla E(x_k) \quad (2.22)$$

which results in descent directions provided $\text{Hess}(x)$ is positive definite. If we assume a quadratic cost function $E = a + b^T x + (x^T A x)/2$, then a necessary and sufficient condition for the minimum is $\nabla E(x) = 0$ ($\text{Hess}(x) > 0$), giving $x = A^{-1}b$ ². When $\gamma=1$ then:

$$\begin{aligned} x_1 &= x_0 - \text{Hess}(x_0)^{-1} \nabla E(x_0) \\ &= x_0 - A^{-1} (b + A x_0) \\ &= -A^{-1} b \end{aligned} \quad (2.23)$$

and thus the minimum can be found in one step. A clear disadvantage of this method is the calculation of the Hessian matrix and its inverse.

2.2 Conjugate descent

Two vectors x_1 and x_2 are said to be mutually conjugate with respect to a symmetric positive definite matrix A , if $x_1^T A x_2 = 0$. In section 2.1 (formula 2.22) it was necessary to calculate the

-
- 1 Only when the minimum corresponds to a hole in the energy surface.
 - 2 This also shows that the problem of solving linear equations is equivalent to the problem of minimizing a quadratic function.

inverse of the Hessian. The conjugate methods are based upon the fact that the inverse of a matrix A can be found by using n mutually conjugate vectors:

$$A^{-1} = \sum_{i=1}^n \frac{x_i x_i^T}{x_i^T A x_i} \quad (2.24)$$

This is proven by showing that $AA^{-1} = I$.

2.2.1 Directional

This method uses a set of descent directions, and uses this set to minimize the objective function without calculating a derivative. Initially they are chosen linearly independent. For each descent direction the optimal γ_k is found by minimizing $E(x_k + \gamma_k s_k)$. After the last descent step from the set has been computed and taken, one mutually conjugate vector is computed and is added to the set of descent directions, while the first direction is deleted from the set. So after n times updating the set, the set will consist of n mutually conjugate directions.

2.2.2 Gradient

The conjugate gradient technique uses the gradient for iteratively finding the set of mutually conjugate vectors.

3 BSB NEURAL NETWORK

In this chapter the various properties of the BSB recall formula are shown. This recall formula is defined as follows: (where (x_{\min}, x_{\max}) , with $(x_{\min} < x_{\max})$, is the vector element space for the bounding hypercube)

$$x_{k+1} = S(x_k + \gamma W x_k) \quad (\gamma > 0) \quad (3.1)$$

$$S(x) = \begin{cases} x_{\max} & x > x_{\max} \\ x & x_{\min} \leq x \leq x_{\max} \\ x_{\min} & x < x_{\min} \end{cases} \quad (3.2)$$

The energy function is defined by:

$$E(x) = -(x^T W x) / 2 \quad (3.3)$$

$E(x)$ is fixed by the learning rule and forms an n-dimensional ellipse. Furthermore, each new vector x_{k+1} represents a point on the energy surface which has less energy than vector x_k . This new vector is found by adding a vector $\Delta x = Wx$, which always is orthogonal to the energy surface, to the old vector. Finally it is shown that when the Hebbian learning rule is used, the resulting weight matrix is positive semi-definite.

3.1 Energy curves

When diagonalizing the weight matrix W used in the energy function E , it is easy to show that E forms an ellipse. Look at the isoclinals of E , which are a set of points x all having the same energy c_1 :

$$\begin{aligned} E(x) &= c_1 \\ -(x^T W x) / 2 &= c_1 \\ x^T W x &= c_2 \\ x^T S A S^{-1} x &= c_2 \\ x^T Q \Lambda Q^T x &= c_2 \\ y^T \Lambda y &= c_2 \\ \lambda_1 y_1^2 + \lambda_2 y_2^2 + \dots + \lambda_n y_n^2 &= c \\ y_1^2 / (1/\lambda_1)^2 + y_2^2 / (1/\lambda_2)^2 + \dots + y_n^2 / (1/\lambda_n)^2 &= c \end{aligned} \quad (3.4)$$

Each $E(x)=c_1$ forms a N dimensional ellipse. The length of the i axis is $2\sqrt{c_1} / \sqrt{\lambda_i}$. Also each axis points towards eigenvector x_i . When W is positive semi-definite E(x) has its maximum at $\nabla E(x)=0$, or $E(x)=0$ for $x=0$. Notice that this is a reversed energy function from the ones studied in chapter 2. This also implies that the minima of E are on an ellipse ¹ which is most far away from $x=0$, and therefore passes through the endpoints of the hypercube formed by formula 3.2. Now we can understand why the BSB model works. During learning vectors are used which represent the endpoints of the hypercube. These vectors are becoming eigenvectors ² of the weightmatrix. During recall these eigenvectors determine the form of the ellipses, which therefore are directed towards the minima of the energy function.

3.2 BSB implements first order gradient descent minimizing energy

The following proof comes from [R.M. Golden 1986]. It shows that the energy associated with x_{k+1} is less than the energy associated with x_k . This implies that subsequent invocations will end up in a vector which has in its local environment minimal energy. Because during learning these minima are associated with prototypes, we may expect to converge to the same prototypes when we start within an environment of each prototype. The gradient of E can be calculated (see appendix A), and denoted as g_k :

$$\nabla E(x_k) = g_k = -Wx_k \quad (3.5)$$

and so, according to formula 2.18 with $s_k = Wx_k$, the BSB recall formula implements a first order gradient descent. We want to investigate the difference of energy Δ between two vectors, and use a difference vector d_k :

$$d_k = x_{k+1} - x_k \quad (3.6)$$

$$\Delta = 2E(x_{k+1}) - 2E(x_k) \quad (3.7)$$

-
- 1 So the minimum does not form a hole. Instead the maximum forms a hill.
 - 2 This depends on the learning rule. With Hebbian learning the learned vectors will only be eigenvectors if they are orthogonal.

The following derivation shows that $\Delta < 0$, thereby proving that BSB tries to minimize energy associated with vectors:

$$\begin{aligned}
 \Delta &= 2E(x_{k+1}) - 2E(x_k) \\
 &= 2E(d_k + x_k) - 2E(x_k) \\
 &= -(d_k + x_k)^T W (d_k + x_k) + x_k^T W x_k \\
 &= -d_k^T W d_k - d_k^T W x_k - x_k^T W d_k \\
 &= -d_k^T W d_k - 2d_k^T W x_k \\
 &= -d_k^T W d_k + 2d_k^T g_k
 \end{aligned} \tag{3.8}$$

$$\text{So } \Delta < 0 \text{ if } d_k^T g_k < (d_k^T W d_k) / 2$$

First look at $d_k^T g_k$. Introduce a function $\alpha(i, k)$ to be able to rewrite formula 3.1 as a linear combination of each vector element:

$$x_{k+1}(i) = x_k(i) - \gamma \alpha(i, k) g_k(i) \tag{3.9}$$

Consider the three regions for x from formula 3.2 and determine the range for $\alpha(i, k)$. The first region:

$$x_{\min} \leq x_k(i) - \gamma g_k(i) \leq x_{\max} \rightarrow \alpha(i, k) = 1 \tag{3.10}$$

The second region:

$$\begin{aligned}
 &x_k(i) - \gamma g_k(i) > x_{\max} \\
 &\left. \begin{aligned} x_{\max} - x_k(i) &< -\gamma g_k(i) \\ x_k(i) &< x_{\max} \end{aligned} \right\} g_k(i) < 0 \\
 &x_{k+1}(i) = x_{\max} = x_k(i) - \gamma \alpha(i, k) g_k(i)
 \end{aligned} \tag{3.11}$$

$$\alpha(i, k) = \frac{x_{\max} - x_k(i)}{-\gamma g_k(i)}$$

$$\rightarrow 0 \leq \alpha(i, k) < 1$$

The third region:

$$\begin{aligned}
 & x_k(i) - \gamma g_k(i) < x_{\min} \\
 & \left. \begin{aligned} x_{\min} - x_k(i) &> -\gamma g_k(i) \\ x_k(i) &> x_{\min} \end{aligned} \right\} g_k(i) > 0 \\
 & x_{k+1}(i) = x_{\min} = x_k(i) - \gamma \alpha(i, k) g_k(i) \quad (3.12) \\
 & \alpha(i, k) = \frac{x_{\min} - x_k(i)}{-\gamma g_k(i)} \\
 & \rightarrow 0 \leq \alpha(i, k) < 1
 \end{aligned}$$

Combining the three regions for $\alpha(i, k)$ leads to:

$$0 \leq \alpha(i, k) \leq 1 \quad (3.13)$$

Now determine $d_k(i)$ and the inproduct:

$$\begin{aligned}
 d_k(i) &= -\gamma \alpha(i, k) g_k(i) \\
 d_k^T g_k &= -\sum (\gamma \alpha(i, k) g_k(i)^2) < 0 \quad (3.14)
 \end{aligned}$$

And look at $(d_k^T W d_k)/2$:

$$\begin{aligned}
 \lambda_{\min} |x|^2 &\leq x^T W x \leq \lambda_{\max} |x|^2 \\
 (\lambda_{\min} |d_k|^2)/2 &\leq (d_k^T W d_k)/2 \quad (3.15)
 \end{aligned}$$

So, when $\lambda_{\min} \geq 0$: $d_k^T g_k < (d_k^T W d_k)/2$. Else, when $\lambda_{\min} < 0$, just calculate the condition for which $\Delta < 0$:

$$\begin{aligned}
 & d_k^T g_k < (d_k^T W d_k)/2 \\
 & -\sum (\gamma \alpha(i, k) g_k(i)^2) < (\lambda_{\min} \sum (-\gamma \alpha(i, k) g_k(i))^2)/2 \quad (3.16) \\
 & \gamma < 2 (\sum \alpha(i, k) g_k(i)^2) / (|\lambda_{\min}| \sum (\alpha(i, k) g_k(i))^2) \\
 & \text{with } \alpha(i, k)^2 \leq \alpha(i, k) \text{ (formula 3.13)} \\
 & \gamma < 2/|\lambda_{\min}|
 \end{aligned}$$

So either when W is positive semi-definite, or $\gamma < 2/|\lambda_{\min}|$, BSB will minimize an energy function.

3.3 Gradient vector orthogonal to energy field

Choose a vector a , with $E(a)=c$. Also place on E a curve $f(t) = (x_1(t), x_2(t), \dots, x_n(t))$, so that for $t=t_0$ $f(t_0)=a$ holds. This leads to the following equality:

$$E(f(t)) = c \quad \forall t$$

$$\frac{dE(x_1(t), x_2(t), \dots, x_n(t))}{dt} = 0 \quad (3.17)$$

$$\nabla E(a)^T f'(t_0) = 0$$

So the gradient vector in a is orthogonal to every derivative of f in a , and so must be orthogonal to the curve $E(x)=c$ through a (derivation from [J.H.J. Almering 1983]).

3.4 Hebbian weightmatrix is positive semi-definite

We want to show that matrix W formed by Hebbian learning is positive semi-definite ($\lambda_{\min} \geq 0$).

All that has to be done is to show that $x^T W x \geq 0 \quad \forall x$ (see appendix A):

$$\begin{aligned} W &= x_1 x_1^T + x_2 x_2^T + \dots + x_n x_n^T \\ x_1^T W x_1 &= x_1^T (x_1 x_1^T + x_2 x_2^T + \dots + x_n x_n^T) x_1 \\ &= x_1^T (x_1^T x_1 x_1 + x_2^T x_1 x_2 + \dots + x_n^T x_1 x_n) \\ &= x_1^T x_1 x_1^T x_1 + x_2^T x_1 x_1^T x_2 + \dots + x_n^T x_1 x_1^T x_n \\ &= (x_1^T x_1)^T x_1^T x_1 + (x_1^T x_2)^T x_1^T x_2 + \dots + (x_1^T x_n)^T x_1^T x_n \\ &= |x_1^T x_1|^2 + |x_1^T x_2|^2 + \dots + |x_1^T x_n|^2 \geq 0 \end{aligned} \quad (3.18)$$

4 BSB RADAR PULSE CLASSIFICATION

A radar receiver converts the received radar spectrum to a number of features or fields for each detected radar. For example the frequency field, the level field and the bearing field. The fields may be distorted with noise. We want two problems to be solved. First, a correct removal of noise, and second, an indication of the number of radars in the neighbourhood.

The radar features are scalars (no dimensionality). A BSB network processes vectors. So a transformation from scalars to vectors is necessary. Two transformations will be discussed. Also results of experiments are presented and explained as much as is possible.

4.1 Translation

A translation translates (converts) a scalar to a vector and vice-versa. Two translation methods will be discussed: thermometer coding and reflected binary coding.

4.1.1 Thermometer coding

A thermometer is formed by a vector. The indicator of the thermometer is a block of vector elements which are ON (have a high value). All other elements are OFF (have a low value). When the scalar has a low value, the indicator of the thermometer is set low (at the beginning, which is the left side, of the vector). Gradually, as the scalar increases, the indicator is set higher (the block of ON vector elements shifts from the beginning of the vector to the end of the vector). Extremes of the scalar need a special treatment, because then the width of the indicator should decrease. A disadvantage of this thermometer coding is that it only uses n code vectors of the 2^n possible vectors in the 2^n hypercube. A reason for choosing this code is that it provides a simple manner to map scalars with slightly differing values to vectors which are close to each other on the hypercube (the thermometer indication of these vectors is almost the same, and so their hamming distance is low, and so they have few differing axis values). The conversion formula, a discussion on a choice for the thermometer width, and the choice for vector elements are presented.

4.1.1.1 Conversion formula

For conversion from scalar to vector the following formula is used: (with s scalar, s_{\max} the maximal field value, s_{\min} the minimal field value, n the number of ON vector elements for that field).

$$\text{mid} = ((s - s_{\min}) / (s_{\max} - s_{\min})) * n \quad (4.1)$$

Test whether there is space at the vector beginning: (with corrlow correction number for a low indication, w width of the block of ON elements and set to a constant value)

$$\begin{aligned} \text{if } (\text{mid} < w/2) \\ \text{corrlow} = (w/2) - \text{mid} \end{aligned} \quad (4.2)$$

Test whether there is space at the vector end to add $w/2$ elements to mid: (with corrhigh correction number for a high indication)

$$\begin{aligned} \text{if } (\text{mid} > n - w/2) \\ \text{corrhigh} = \text{mid} + (w/2) - n \end{aligned} \quad (4.3)$$

Now mid is used as index to the x vector, around mid a total of w values are set to x_{\max} , unless mid is at a border: (with x_{\min} the minimal value of a vector element, x_{\max} the maximal value of a vector element, fi the first index which must be set in the vector to x_{\max})

$$\begin{aligned} fi &= \text{mid} + \text{corrlow} - w/2 \\ \text{for } (i = 0; i < fi; i++) & \quad x[i] = x_{\min} \\ \text{for } (i = fi; i < fi + w - \text{corrlow} - \text{corrhigh}; i++) & \quad x[i] = x_{\max} \\ \text{for } (i = fi + w - \text{corrlow} - \text{corrhigh}; i < n; i++) & \quad x[i] = x_{\min} \end{aligned} \quad (4.4)$$

As an example of a thermometer coding look at table 4.1.

0	1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1.5
2	-1	1	1	1	-1	-1	-1	-1	-1	-1	-1	2.5
3	-1	-1	1	1	1	-1	-1	-1	-1	-1	-1	3.5
4	-1	-1	-1	1	1	1	-1	-1	-1	-1	-1	4.5
5	-1	-1	-1	-1	1	1	1	-1	-1	-1	-1	5.5
6	-1	-1	-1	-1	-1	1	1	1	-1	-1	-1	6.5
7	-1	-1	-1	-1	-1	-1	1	1	1	-1	-1	7.5
8	-1	-1	-1	-1	-1	-1	-1	1	1	1	-1	8.5
9	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	9.5
10	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	10.0

Table 4.1 Thermometer coding with $s_{\min}=0$, $s_{\max}=11$, $x_{\min}=-1$, $x_{\max}=1$, $w=3$, $n=11$. The first column gives scalar input, the second column vector values, the last column scalar output.

For the conversion from vector to scalar the following formula is used: (with val value of first element of block of thermometer indication, w the number of ON vector elements the block consists of, s_{\max} the maximal field value, s_{\min} the minimal field value, n the number of vector elements for that field)

$$s = s_{\min} + (val + w/2) * (s_{\max} - s_{\min}) / n \quad (4.5)$$

4.1.1.2 Choice of thermometer width

The choice of thermometer width used with the thermometer coding has consequences for the separation of vectors during recalling. If we use Hebbian learning, only orthogonal vectors can be recalled correctly:

$$\begin{aligned} W &= x_1 x_1^T + x_2 x_2^T + \dots + x_l x_l^T + \dots + x_n x_n^T \\ W x_j &= x_1 x_1^T x_j + x_2 x_2^T x_j + \dots + x_l x_l^T x_j + \dots + x_n x_n^T x_j \\ &= x_1^T x_j x_1 + x_2^T x_j x_2 + \dots + x_l^T x_j x_l + \dots + x_n^T x_j x_n \\ &= |x_j| |x_j| \text{ provided } x_i^T x_j = 0 \text{ if } i \neq j \end{aligned} \quad (4.6)$$

When vectors are formed according to the (-1,1) thermometer coding, they generally will not be orthogonal. This is not as bad as it may seem, because thanks to gradient descent recalling, non-orthogonal vectors can be separated too. In fact the setting of w determines the level of orthogonality between input vectors. For example consider two vectors y_1 and y_2 consisting each of two fields of 11 elements. Give both fields of y_1 a low thermometer coding and both fields of y_2 a high coding, and calculate the inproduct of $y_1^T y_2$, varying the value of w:

When $w=1$:

$$y_1 = (-1 \ 1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ 1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1)^T$$

$$y_2 = (-1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ 1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ 1 \ -1)^T$$

the inproduct $y_1^T y_2 = 14$.

When $w=3$:

$$y_1 = (1 \ 1 \ 1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ 1 \ 1 \ 1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1)^T$$

$$y_2 = (-1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ 1 \ 1 \ 1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ 1 \ 1 \ 1)^T$$

the inproduct $y_1^T y_2 = -2$.

When $w=4$:

$$y_1 = (1 \ 1 \ 1 \ 1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ 1 \ 1 \ 1 \ 1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1)^T$$

$$y_2 = (-1 \ -1 \ -1 \ -1 \ -1 \ -1 \ 1 \ 1 \ 1 \ 1 \ -1 \ -1 \ -1 \ -1 \ -1 \ 1 \ 1 \ 1 \ 1)^T$$

the inproduct $y_1^T y_2 = -10$.

Varying w results in varying the level of orthogonality. When we demand that non-correlated vectors are orthogonal, we can derive the optimal value of w :

$$\begin{aligned} y_1 &= (1_1 \dots 1_w \ -1_{w+1} \dots \dots \dots -1_n)^T \\ y_2 &= (-1_1 \dots \dots \dots -1_{n-w} \ 1_{n-w+1} \dots \dots 1_n)^T \\ y_1^T y_2 &= (-w) + (n-w+1 - (w+1)) + (-w) \\ &= n - 4w \\ w &= n/4 \end{aligned} \quad (4.7)$$

This choice of w leads to a maximum of only 4 orthogonal vectors, independent of n .

4.1.1.3 Choice for value of vector elements

In principle two codings are possible: (0,1) coding and (-1,1) coding. It is shown that the (0,1) coding results in W matrices which are band matrices and so prevent some prototype recollection (there is no information outside the bands), resulting in the choice for (-1,1) coding.

Consider a vector y with thermometer width $w=3$ consisting of 10 elements (0,1) coded with a block of ones starting at index 4, $y = (0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0)^T$, and look at W :

$$W = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} (0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0)^T$$

$$= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

By varying for different y_s the value of the start of indication but keeping $w=3$, the block of ones in W shifts across the diagonal, and so the next W results: (each x denotes a matrix element which may have a value $\neq 0$)

$$W = \begin{pmatrix} x & x & x & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ x & x & x & x & 0 & 0 & 0 & 0 & 0 & 0 \\ x & x & x & x & x & 0 & 0 & 0 & 0 & 0 \\ 0 & x & x & x & x & x & 0 & 0 & 0 & 0 \\ 0 & 0 & x & x & x & x & x & 0 & 0 & 0 \\ 0 & 0 & 0 & x & x & x & x & x & 0 & 0 \\ 0 & 0 & 0 & 0 & x & x & x & x & x & 0 \\ 0 & 0 & 0 & 0 & 0 & x & x & x & x & x \\ 0 & 0 & 0 & 0 & 0 & 0 & x & x & x & x \end{pmatrix}$$

Now take a vector y_s of n elements and width w : (with s the index of the first ON bit)

$$y_s = (0_1 \dots 0_{s-1} \ 1_s \dots 1_{s+w-1} \ 0_{s+w} \dots 0_n)^T \quad (4.8)$$

Matrix W is formed by summing $y_s y_s^T$ for $1 \leq s \leq n$. Look at $W y_s$ (choose one s). The most extreme elements of the block (1_s and 1_{s+w-1}) can activate a subblock in $W y_s$:

$$W y_s = (0_1 \dots 0_{s-w} \ 1_{s-w+1} \dots 1_{s+2w-2} \ 0_{s+2w-1} \dots 0_n)^T \quad (4.9)$$

and $W y_s$ is always limited to a band of $3w-2$.

4.1.2 Reflected binary coding

The thermometer code only uses n nodes of the 2^n hypercube as possible code vectors. The reflected binary code (also called Gray code) uses each node [A.P. Thijssen, H.A. Vink, C.H.

Eversdijk 1982]. Also scalar values which differ slightly are translated into vectors which have a low hamming distance.

Using the thermometer code, the hamming distance between two vectors, each representing a scalar, increases linearly with the increase of the difference between the scalars. Using reflected binary coding the hamming distance between two vectors representing two differing scalars can also increase if the difference between the scalars increases, but this does not happen linearly. The distance between the vectors representing scalars 0 and 1, $d(0,1)$ is 1, as can be seen in table 4.2. It can also be seen that:

$$\begin{aligned} d(0,1) &= d(0,3) \\ d(0,2) &= d(0,4) \\ d(0,2) &> d(0,3) \end{aligned} \quad (4.10)$$

4.1.2.1 Conversion formula

For conversion from scalar to vector the following formula is used: (with $0 \leq s < 2^n$ scalar, r intermediate number, $x = x_{n-1} \dots x_0$ vector)

$$\begin{aligned} r &= s \bmod (2^i) \quad (i = 2, 3, \dots, n+1) \\ \text{if } (r < 2^{i-2}) \quad x_{i-2} &= x_{\min} \\ \text{if } (2^{i-2} \leq r < 2^{i-2} + 2^{i-1}) \quad x_{i-2} &= x_{\max} \\ \text{if } (r \geq 2^{i-2} + 2^{i-1}) \quad x_{i-2} &= x_{\min} \end{aligned} \quad (4.11)$$

An example of Gray coding is shown in table 4.2.

0	-1	-1	-1	-1
1	-1	-1	-1	1
2	-1	-1	1	1
3	-1	-1	1	-1
4	-1	1	1	-1
5	-1	1	1	1
6	-1	1	-1	1
7	-1	1	-1	-1
8	1	1	-1	-1
9	1	1	-1	1
10	1	1	1	1

Table 4.2 Gray coding with $s_{\min}=0$, $s_{\max}=11$, $x_{\min}=-1$, $x_{\max}=1$, $n=4$. The first column gives scalar input (is scalar output), the second column gives vector values.

For reverse conversion the following formula is used: (with $x = x_{n-1} x_{n-2} \dots x_0$ the reflected binary coded vector, $s_{n-1} s_{n-2} \dots s_0$ the resulting binary coded vector, s the corresponding scalar)

$$\begin{aligned}
 s_{n-1} &= x_{n-1} \\
 \text{if } (s_{i+1} = x_{\max}) \quad (i &= n-2, n-3, \dots, 0) \\
 \quad \text{if } (x_i = x_{\min}) \quad s_i &= x_{\max} \\
 \quad \text{else} \quad s_i &= x_{\min} \\
 \text{else } s_i &= x_i
 \end{aligned} \tag{4.12}$$

$$b(s_i) = \begin{cases} 1 & \text{if } s_i = x_{\max} \\ 0 & \text{if } s_i = x_{\min} \end{cases}$$

$$s = b(s_{n-1})2^{n-1} + b(s_{n-2})2^{n-2} + \dots + b(s_0)2^0$$

A few remarks on the learning algorithm which is used. When hebbian learning is used, the input patterns should be almost orthogonal. This limits the network to n distinguishable vectors. When delta learning is used, every set of n linearly independent vectors can be used. So although the 2^n hypercube has 2^n corners and though the reflected binary code uses each of them, only n of them can actually be associated with a prototype with the standard BSB model. An extension might be to develop a new BSB model with hidden units and back propagation learning process. In fact only $n-1$ linearly independent vectors can be learned by an auto-associative system, because if n vectors are learned so that $Wx_i = \lambda x_i$ ($i = 1, 2, \dots, n$), you have fully specified the set of eigenvectors, so the only solution is $W = I$ (see also section 4.2.2).

4.1.2.2 Choice of value of vector elements

Again the (0,1) coding is not a good coding. With the reflected binary code not each vector has the same amount of ON vector elements. When a vector with few ON vector elements is learned and a vector with many ON vector elements, the first vector will get a smaller eigenvalue (determined by the length of the input vector), and will therefore be learned much weaker.

4.2 Results of experiments

First a simple 2 dimensional BSB model is used to verify the theories developed. Then an experiment is done which tests the classification possibilities of the BSB network.

4.2.1 2 dimensional BSB

In this section the experiment that Anderson published in [J.A. Anderson 1983] is repeated. The purpose is to study the recall field of a 2 dimensional BSB model. This field is a 2 dimensional plane, bounded by $x_{\min}=-1$ and $x_{\max}=1$ of the vector elements. The weightmatrix is:

$$W = \begin{vmatrix} 0.035 & -0.005 \\ -0.005 & 0.035 \end{vmatrix}$$

In figure 4.1 this recall field is shown. In table 4.3 the number of steps needed for convergence is given for each gridpoint. Because $W > 0$, γ can be chosen large to reduce the number of steps. See table 4.4.

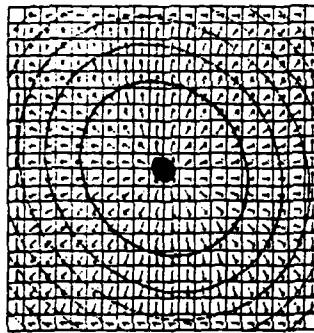


Figure 4.1 Recall field for a 2 dimensional BSB model. In each gridpoint the gradient is drawn.

Figure 4.1 also shows the ellipses $E(x)=c$ for different values of c ($c=0$, $c=-0.005$, $c=-0.01$, ...). The gradient vectors indeed are orthogonal to these ellipses.

0	4	9	14	20	28	39	58	118	74	52	39	31	24	18	14	10	6	3	0
4	4	9	14	20	28	39	57	116	75	52	39	31	24	18	14	10	6	3	3
9	9	8	14	20	28	39	57	111	76	52	40	31	24	19	14	10	7	6	6
14	14	14	13	20	27	38	55	105	77	53	40	31	24	19	14	10	10	10	10
20	20	20	20	19	27	37	54	97	80	54	41	32	25	19	14	14	14	14	14
28	28	28	27	27	26	36	52	89	84	56	42	32	25	20	19	19	19	18	18
39	39	39	38	37	36	34	49	80	91	58	43	33	26	25	25	24	24	24	24
58	57	57	55	54	52	49	46	72	103	61	45	35	33	32	32	31	31	31	31
118	116	111	105	97	89	80	72	63	140	66	48	45	43	42	41	40	40	39	39
74	75	76	77	80	84	91	103	140	100	76	66	61	58	56	54	53	52	52	52
52	52	52	53	54	56	58	61	66	76	100	140	103	91	84	80	77	76	75	74
39	39	40	40	41	42	43	45	48	66	140	63	72	80	89	97	105	111	116	118
31	31	31	31	32	32	33	35	45	61	103	72	46	49	52	54	55	57	57	58
24	24	24	24	25	25	26	33	43	58	91	80	49	34	36	37	38	39	39	39
18	18	19	19	19	20	25	32	42	56	84	89	52	36	26	27	27	28	28	28
14	14	14	14	14	19	25	32	41	54	80	97	54	37	27	19	20	20	20	20
10	10	10	10	14	19	24	31	40	53	77	105	55	38	27	20	13	14	14	14
6	6	7	10	14	19	24	31	40	52	76	111	57	39	28	20	14	8	9	9
3	3	6	10	14	18	24	31	39	52	75	116	57	39	28	20	14	9	4	4
0	3	6	10	14	18	24	31	39	52	74	118	58	39	28	20	14	9	4	0

Table 4.3 Number of steps necessary for convergence with $\gamma=1$.

The values from table 4.3 perhaps will be better understood when they are put in a landscape plot, where the height of the landscape on grid coordinate (x,y) is determined by the number of steps for (x,y) to converge. See figure 4.2.

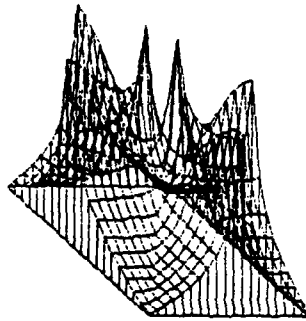


Figure 4.2 Table 4.3 as a landscape.

[illegible]

4.2.2 Clustering

For the experiments the following parameters are used. The neural network consists of 22 completely linked neurons. Each radar signal consists of two fields. Each experiment is done four times. First each field is thermometer coded with a thermometer width of 3 and both Hebbian and delta learning is used, then each field is Gray coded and both Hebbian and delta learning is used. Each field serves as input for 11 neurons ($w=3$, $n=11$ for each field). During recalling each possible inputvector in the inputrange is generated and the stable outputvector belonging to these inputvectors is drawn in a two-dimensional plot. For simplicity both x and y field values are between 0 and 11 (so (0,0) corresponds with the upper left corner, (10,10) with the lower right corner, $s_{\max}=11$, $s_{\min}=0$). When delta learning is used, L_{coef} is chosen 0.05, and each pattern is learned 5 times, after which randomly another pattern is learned 5 times. This is repeated until all patterns have been learned.

4.2.2.1 One radar

The radar learned is at $(1,1)$ ¹. Look at figure 4.3 for the recall diagram. In this figure and in all following figures the arrows in each gridpoint (x,y) point to the stable gridpoint (v,w) to which (x,y) converges when formula 3.1 is applied recursively.

1 Recall values will be at (1.5, 1.5), see table 4.1.

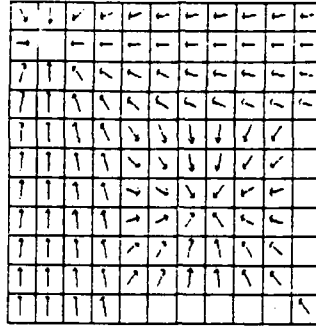


Figure 4.3 Recall diagram for one radar consisting of two fields, learned at (1,1) using thermometer coding.

Points (.) in the plot represent vectors of which the recall vector equals the original vector. There are two kinds of these vectors. First the radar point (1,1), which is learned to be an eigenvector, and so $Wx=\lambda$, with $\lambda=22$. Because only one vector is learned, there will be 21 eigenvectors with $\lambda=0$ ($\lambda=0$ has multiplicity 21). This implies that there exists a 21 dimensional plane on which vectors are lying with $Wx=0$. These vectors form the second group equilibrium points. The energy of these vectors is $E(x) = -(x^T W x)/2 = -(x^T 0)/2 = 0$. The diagram also shows another interesting feature. Although no radar is learned at (7,7), a lot of arrows are pointing towards this point. Again the explanation is simple. When $Wx = \lambda x$, then also $W(-x) = -Wx = -\lambda x = \lambda(-x)$, or when x is an eigenvector then also $-x$ is an eigenvector. We have learned the following vectors:

$$\begin{aligned} x = (1,1) &= (1 \ 1)^T \\ -x &= (-1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1)^T \end{aligned}$$

The radar corresponding to $-x$ will be: (with formula 4.5)

$$\begin{aligned} \text{rad} &= s_{\min} + (val + w/2) * (s_{\max} - s_{\min}) / n \\ &= (3 + (8/2) * 11/11, 3 + (8/2) * 11/11) \\ &= (7, 7) \end{aligned} \tag{4.13}$$

Also notice that the energy of eigenvector $-x$ equals:

$$\begin{aligned} E(x) &= -(x^T W x) / 2 \\ E(-x) &= -((-x)^T W (-x)) / 2 \\ &= E(x) \end{aligned} \tag{4.14}$$

This leads to two remarks, concerning the fact that we don't want to reach equilibrium points which are not a result of x (of learned prototypes). First, it is, for this example, easy to escape from $Wx=0$ points. Because $Ex=0$, these x points will be located at the top of the energy landscape, and when one element of x is changed so that $E(x) \neq 0$, x will converge to a learned x . In fact this implements simulated annealing [P.J.M. Laarhoven, E.H.L. Aarts 1988] (x is given a little push, so it rolls down from the top of the hill). For the second group of equilibrium vectors, the obvious solution is to invert the vector when the recalled thermometer width is greater than the expected thermometer width.

Greenberg [H.J. Greenberg 1988] has studied equilibria of the BSB neural model. One result is that when W is strongly diagonal-dominant², each extreme point (endpoint of hypercube) is stable. It can be proven that, when starting in the neighbourhood of each extreme point, BSB will converge to the extreme point. The other result is that when W is strongly diagonal-dominant, and x is a fixed point which is no extreme, then x is not stable. This occurs when $Wx=0$.

In figure 4.4 the recall field is shown when Gray coding is used. Each input vector is recalled perfectly.

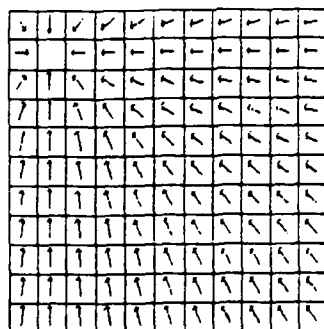


Figure 4.4 Recall diagram for one radar (1,1) consisting of two fields, using Gray coding.

² W is strongly diagonal-dominant when each diagonal element is larger than the sum of the rest of the elements of the row the diagonal element belongs to.

4.2.2.2 Two radars

Two radars are learned, at (1,1) and (9,9). Look at figure 4.5 for the recall diagrams.

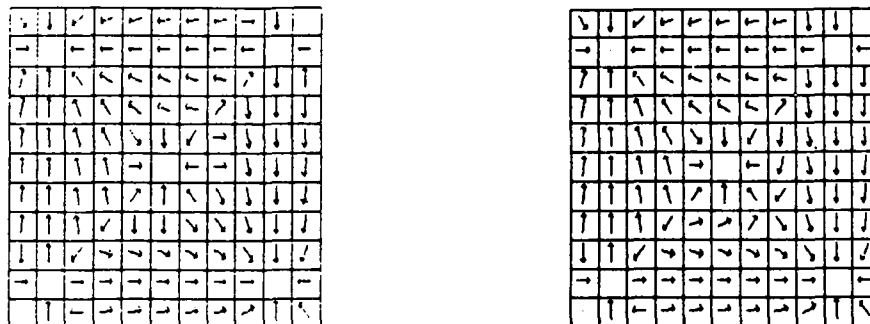


Figure 4.5 Recall diagram for two radars (1,1), (9,9), using thermometer coding and Hebbian learning (left) or delta learning (right).

As can be expected, the region above the diagonal converges to (1,1) and the region below the diagonal to (9,9). On the diagonal are equilibrium points. These extra points are (0,10), (1,9), (5,5), (9,1) and (10,0), both when Hebbian and delta learning is used. In each of these points the gradient vector has the direction: (This is only checked for Hebbian learning)

$$(0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0)^T.$$

The gradient has a zero component in the direction of each prototype. This results in the vector:

$$(-1 \ -1 \ -1 \ 1 \ 1 \ 1 \ 1 \ 1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ 1 \ 1 \ 1 \ 1 \ 1 \ -1 \ -1 \ -1)^T,$$

or scalar (5.5, 5.5).

See figure 4.6 for the recall field when translation is done with Gray coding. It appears that (1,9), (6,6) and (9,1) are extra equilibrium points when Hebbian learning is used.

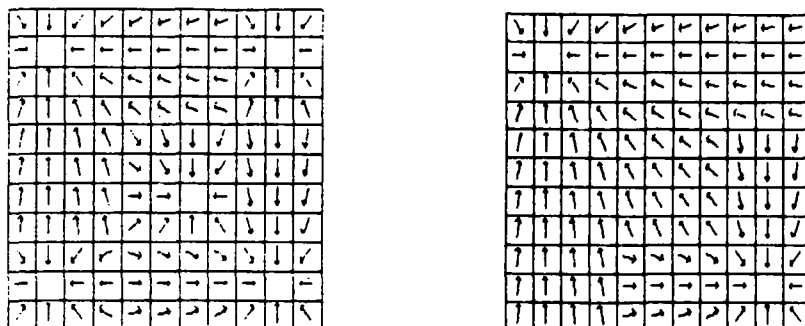


Figure 4.6 Recall diagram for two radars (1,1), (9,9), using Gray coding and Hebbian learning (left) or delta learning (right).

From table 4.2 it can be derived that for $s=4, 5, 6$ and 7 the corresponding vector has the same Hamming distance to $s=1$ and $s=9$. When a field has one of these values, the output is $s=6$, which has the *minimal* Hamming distance 1 to $s=1$ and $s=9$. When delta learning is used, the extra equilibrium points have disappeared and are replaced by convergence to the radar (1,1).

4.2.2.3 3 radars

The third radar is placed at (8,8). Figure 4.7 shows that the diagonal has moved towards the upper left corner both for Hebbian and delta learning. The gradient is: (checked only for Hebbian learning)

$$(42 \ 42 \ 42 \ 30 \ 30 \ 30 \ 30 \ -6 \ -42 \ -42 \ -6 \ 42 \ 42 \ 42 \ 30 \ 30 \ 30 \ 30 \ -6 \ -42 \ -42 \ -6)^T$$

This gives rise to the vector

$$(1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ -1 \ -1 \ -1 \ -1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ -1 \ -1 \ -1 \ -1)^T$$

or (3.5, 3.5).

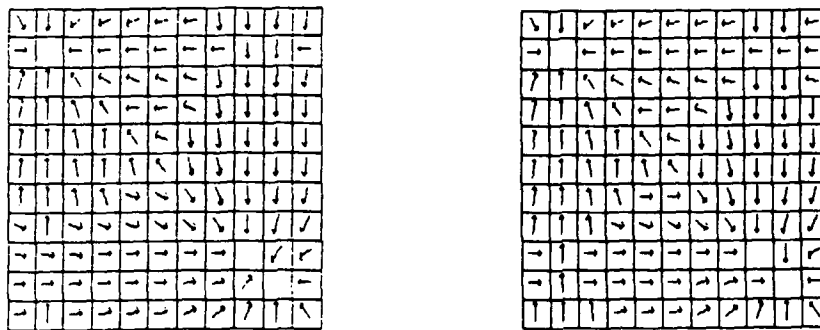


Figure 4.7 Recall diagram for 3 radars (1,1), (9,9), (8,8), using thermometer coding and Hebbian learning (left) or delta learning (right).

Figure 4.8 shows the recall diagram for Gray coding. Only one stable point (9,9) exists when Hebbian learning is used. Inspection of the weight matrix (for one field):

$$W = \begin{pmatrix} 3 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & 3 & -1 & -1 & 3 & 3 & 3 & 3 & 3 & 3 \\ -1 & -1 & 3 & 3 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & 3 & 3 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & 3 & -1 & -1 & 3 & 3 & 3 & 3 & 3 & 3 \\ -1 & 3 & -1 & -1 & 3 & 3 & 3 & 3 & 3 & 3 \\ -1 & 3 & -1 & -1 & 3 & 3 & 3 & 3 & 3 & 3 \\ -1 & 3 & -1 & -1 & 3 & 3 & 3 & 3 & 3 & 3 \\ -1 & 3 & -1 & -1 & 3 & 3 & 3 & 3 & 3 & 3 \\ -1 & 3 & -1 & -1 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

So each Wx will have direction $(1 \ -1 \ 1 \ 1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1)^T$, or scalar 9.

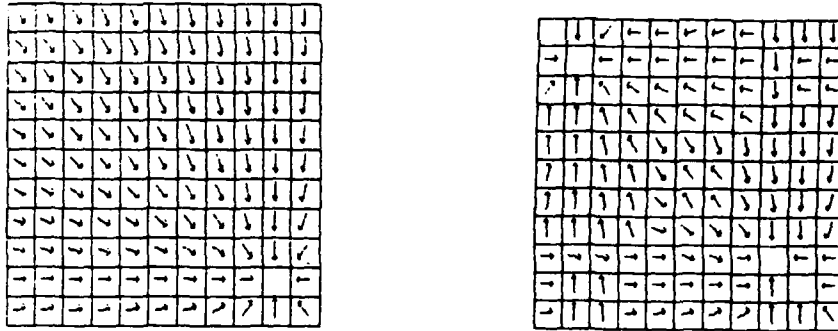


Figure 4.8 Recall diagram for 3 radars (1,1), (9,9), (8,8), using Gray coding and Hebbian learning (left) or delta learning (right).

4.2.2.4 4 radars

Figures 4.9 and 4.10 show recall fields when 4 radars are learned. These diagrams correspond with the diagrams of 2 learned radars.

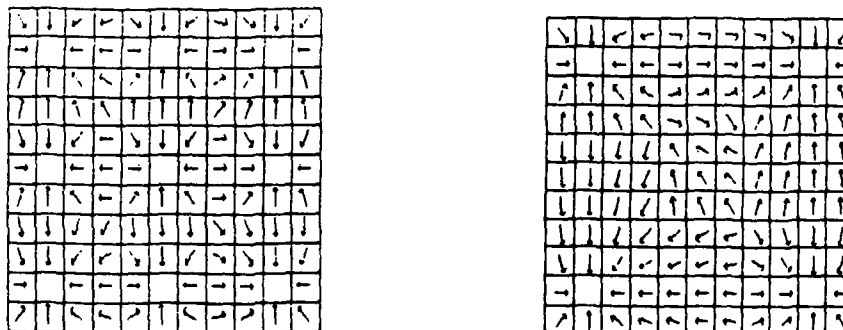


Figure 4.9 Recall diagram for 4 radars (1,1), (1,9), (9,1), (9,9), using thermometer coding and Hebbian learning (left) or delta learning (right).

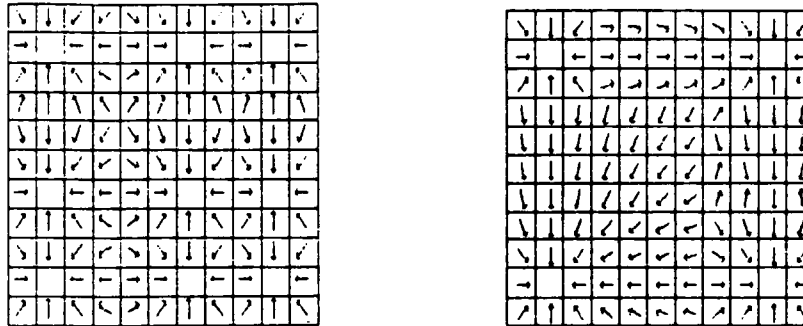


Figure 4.10 Recall diagram for 4 radars (1,1), (1,9), (9,1), (9,9), using Gray coding and Hebbian learning (left) or delta learning (right).

When Hebbian learning is used, extra stable equilibrium points are formed on each separation line between two radars. With delta rule learning these extra points also converge to a learned radar³.

4.2.2.5 How many radars can be learned?

For the last experiment it is verified whether a network of n neurons can learn $n-1$ prototypes. The network used consisted of 8 completely linked neurons for one field. Only delta rule learning is used, and Gray coding. The following seven prototypes are learned: 12, 20, 45, 63, 128, 211 and 233. Lcoef is 0.05 and each pattern learned 7 times. The following weight matrix is formed:

$$W = \begin{bmatrix} 7.6 & -0.8 & -0.0 & 1.2 & 0.0 & 0.4 & 0.8 & -0.4 \\ -0.8 & 6.4 & -0.0 & 2.4 & 0.0 & 0.8 & 1.6 & -0.8 \\ -0.0 & -0.0 & 8.0 & -0.0 & 0.0 & -0.0 & 0.0 & -0.0 \\ 1.2 & 2.4 & -0.0 & 4.4 & 0.0 & -1.2 & -2.4 & 1.2 \\ 0.0 & 0.0 & 0.0 & 0.0 & 8.0 & 0.0 & -0.0 & 0.0 \\ 0.4 & 0.8 & -0.0 & -1.2 & 0.0 & 7.6 & -0.8 & 0.4 \\ 0.8 & 1.6 & 0.0 & -2.4 & -0.0 & -0.8 & 6.4 & 0.8 \\ -0.4 & -0.8 & -0.0 & 1.2 & 0.0 & 0.4 & 0.8 & 7.6 \end{bmatrix}$$

It can be verified that the seven prototypes are indeed eigenvectors of this matrix, with $\lambda=8$ for each vector.

3 To which prototypes they are converging deserves further attention, because the chosen prototype appears to be not always a fair one.

Now prototype number eight (with a value of 254) is added to the set, and the whole set is learned again (starting with $W=0$). The weight matrix which is formed: $W=8I$. So the system is not able to form the n^{th} eigenvector.

Looking again at the weight matrix formed by learning the seven prototypes, shows that it forms a strongly diagonal-dominant matrix. This implies that each hypercube corner is stable. This has the consequence that the BSB recall will not converge to one of the prototypes, but instead remain on each stable hypercube corner.

5 CONCLUSIONS AND FURTHER RESEARCH

The BSB neural network implements a first order gradient descent algorithm, capable of solving nonlinear optimization problems such as radar pulse classification. A first step has been done towards the implementation of radar pulse classification with a BSB network.

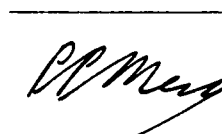
The experiments show that the BSB network is able to remove noise from signals, if the original signals are learned in a separate learning phase. A network of n neurons can learn $n-1$ radars effectively. Noisy input can be recalled by the BSB network to one of these $n-1$ radars, if the weight matrix is not diagonal-dominant.

When there is no separate learning phase, learning and recalling can be combined. A number of radar data samples is processed, after which the number of stable corners indicates the number of different radars (clusters). Further research is required to study and implement this possibility.

The BSB neural network can also be used in other applications. TNO-FEL has studied and implemented a Spatio-temporal Pattern Recognition (SPR) neural network to recognize garbled text strings in messages [P.P. Meiler 1990]. This recognition can also be done using a BSB neural network. Each character of the string is represented by a field using 5 bit Gray coding. This approach may be more efficient than the SPR network because there is no time sequence involved. However, the number of iterations required for the BSB network to converge to the correct result is as yet unknown. A comparison may give interesting results.



P. van Lieshout
(Group leader)



P.P. Meiler
(Author)



A. van Wezenbeek
(Author)

6 REFERENCES

- Almering J.H.J., *Analyse*, Delftse Uitgevers Maatschappij, 1983
- Anderson J.A., *Neural models with cognitive implications*, in: Basic processes in reading perception and comprehension, edited by D. Laberge, S.J. Samuels, Hillsdale, NJ: Erlbaum, 1977, 27-90
- Anderson J.A., *Cognitive and psychological computation with neural models*, IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-13(5), September/October 1983, 799-815
- Anderson J.A., *Radar signal categorization using a neural network*, INNS Boston 1988
- Anderson J.A., Hinton G.E., *Models of information processing in the brain*, in: Parallel models of associative memory, edited by G.E. Hinton, J.A. Anderson, Lawrence Erlbaum Associates Publishers, 1981, 9-48
- Anderson J.A., Mozer M.C., *Categorization and selective neurons*, Parallel models of associative memory, edited by Hinton G.E., Anderson J.A., Lawrence Erlbaum Associates Publishers, 1981, 213-236
- Golden R.M., *The Brain-State-in-a-Box neural model is a gradient descent algorithm*, Journal of Mathematical Psychology 30, 1986, 73-80
- Greenberg H.J., *Equilibria of the Brain-State-in-a-Box neural model*, Neural Networks, Vol. 1, 1988, 323-324
- Jacoby S.L.S., Kowalik J.S., Pizzo J.T., *Iterative methods for nonlinear optimization problems*, Prentice-Hall Inc., 1972
- Laarhoven P.J.M. van, Aarts E.H.L., *simulated annealing: theory and applications*, D. Reidel Publishing Company 1988
- Meiler P.P., *Garbled text string recognition with a spatio-temporal pattern recognition neural network*, FEL-90-B131, TNO-FEL, 1990
- Rumelhart D.E., McClelland J.L., *Parallel distributed processing: explorations in the micro-structure of cognition, Vol.1 and Vol. 2*, MIT Press, 1987
- Sorenson H.W., *Parameter estimation: principles and problems*, Control and systems theory, Vol. 9, Marcel Dekker Inc., 1980
- Strang G., *Linear algebra and its applications*, Academic Press 1980
- Thijssen A.P., Vink H.A., Eversdijk C.H., *Digitale Techniek 1*, Delftse Uitgevers Maatschappij 1982

LINEAR ALGEBRA

1 Eigenvalues and eigenvectors

Eigenvalues and eigenvectors form one of the most important concepts of linear algebra. In this section it is shown how a matrix A can be diagonalized, thus revealing its eigenvalues and eigenvectors. The derivation comes from [G. Strang 1980].

$Ax = \lambda x$, λ eigenvalue, x eigenvector

$$\begin{aligned}
 AS &= A \begin{bmatrix} s_1 & \dots & s_n \end{bmatrix}, S \text{ matrix of eigenvectors} \\
 &= \begin{bmatrix} \lambda_1 s_1 & \dots & \lambda_n s_n \end{bmatrix} \\
 &= \begin{bmatrix} \lambda_1 s_{11} & \dots & \lambda_n s_{n1} \\ \lambda_1 s_{12} & \dots & \lambda_n s_{n2} \\ \vdots & & \vdots \\ \lambda_1 s_{1n} & \dots & \lambda_n s_{nn} \end{bmatrix} \\
 &= \begin{bmatrix} s_{11} & \dots & s_{n1} \\ s_{12} & \dots & s_{n2} \\ \vdots & & \vdots \\ s_{1n} & \dots & s_{nn} \end{bmatrix} \times \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \dots & 0 & \lambda_n \end{bmatrix} \\
 &= S \times \Lambda
 \end{aligned}$$

$$ASS^{-1} = SAS^{-1}$$

$$A = SAS^{-1}$$

We now look at the matrix of orthonormal eigenvectors Q

$$\begin{aligned}
 Q^T Q &= \begin{bmatrix} q_1^T \\ q_2^T \\ \vdots \\ q_n^T \end{bmatrix} \times \begin{bmatrix} q_1 & q_2 & \dots & q_n \end{bmatrix} \\
 Q^T Q &= I \\
 Q^{-1} &= Q^T
 \end{aligned}$$

$$\begin{aligned}
A &= Q\Lambda Q^T \\
x^T A x &= x^T Q\Lambda Q^T x \\
&= y^T \Lambda y \quad (\text{choose } y = Q^T x) \\
&= |y_1 \ y_2 \ \dots \ y_n| \times \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \lambda_n \end{bmatrix} \times \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \\
&= |y_1 \ y_2 \ \dots \ y_n| \times \begin{bmatrix} \lambda_1 y_1 \\ \lambda_2 y_2 \\ \vdots \\ \lambda_n y_n \end{bmatrix} \\
&= \lambda_1 y_1^2 + \lambda_2 y_2^2 + \dots + \lambda_n y_n^2 \\
\rightarrow \lambda_{\min} |y|^2 &= \lambda_{\min} |x|^2 \leq x^T A x \leq \lambda_{\max} |y|^2 = \lambda_{\max} |x|^2
\end{aligned}$$

A matrix A is called positive semi-definite when all eigenvalues are greater than or equal to zero, so $\lambda_{\min} \geq 0$, or $x^T A x \geq 0$. When $\lambda_{\min} > 0$, so $x^T A x > 0$, A is called positive definite.

2 Linear operators

In this section a summary is given of elementary differential operators [H.W. Sorenson 1980].

The first order derivative of f (written as a row vector):

$$\begin{aligned}
\frac{\delta f}{\delta x} &= \left| \frac{\delta f}{\delta x_1} \quad \dots \quad \frac{\delta f}{\delta x_n} \right|^T \\
&= \nabla f
\end{aligned}$$

The second order derivative of f , also called Hessian (or Hess), forming a $(n \times n)$ matrix:

$$\begin{aligned}
\frac{\delta^2 f}{\delta x^2} &= \frac{\delta}{\delta x} \frac{\delta f}{\delta x} \\
&= \frac{\delta}{\delta x} \nabla f
\end{aligned}$$

The following derivatives are used:

$$\begin{aligned}
\frac{\delta (c^T x)}{\delta x} &= c^T \\
\frac{\delta (Ax)}{\delta x} &= A
\end{aligned}$$

The chain rule:

$$\frac{\delta f(u)}{\delta x} = \frac{\delta f}{\delta u} \frac{\delta u}{\delta x}$$

As an example the chain rule can be used to find the derivative of $f = x^T A x$:

$$f = x^T A x$$

$$g(x, y) = x^T y, \quad y = A x$$

$$\begin{aligned} \frac{\delta f}{\delta x} &= \frac{\delta g}{\delta x} + \frac{\delta g}{\delta y} \frac{\delta y}{\delta x} \\ &= y^T + x^T A \\ &= x^T A^T + x^T A \\ &= x^T (A + A^T) \end{aligned}$$

$$\text{if } A = A^T, \text{ then } \nabla f(x) = 2Ax.$$

Second order Taylor series approximation in the neighbourhood of x :

$$f(x+h) \approx f(x) + h^T \nabla f(x) + (h^T \text{Hess}(x) h) / 2$$

DOCUMENTATION OF C-SOURCES

1 Short description of each file

The name of each C file is an abbreviation of its function. This function operates on data. Data has a certain format and a certain type.

The ASCII format represents readable numbers (integers or floats). Binary format represents bytes.

Scalar type indicates that the data must be interpreted as single entities. Vector type indicates that the data forms a sequence of vectors. Matrix type indicates that the data forms just one matrix. To use the BSB network for garbled string recognition also String type can be chosen

When the function specifies a conversion, it converts input with a certain format and type to output with a certain format and type. So a good mnemonic for a conversion file should give an indication of the formats and types it expects for its in- and output. The first letter of characterizing format and type is a good choice (A = ascii, B = binary, S = scalar, V = vector, M = matrix, S = string). Unfortunately this gives rise to too long mnemonic names (at least 4 letters). Each name is limited to 3 letters with the addition of the full name it actually stands for.

Finally for each file is given whether it contains a main program, and if so the name of the executable it generates, or a routine. Common parameters for all programs (such as the number of neurons) are read from one file BSB.PAR.

- 2D: main for 2D, creates a two dimensional plot consisting of a field of arrows, optionally it also generates a memory dump (picture file) or invokes PRNT to generate a print file
- 2Dpl: routine 2Dpl, does the real plot work
- 3D: main for 3D, creates a three dimensional plot, z-axis values forming the height of a x-y plane, optionally it also generates a memory dump (picture file) or invokes PRNT to generate a print file
- ACB: routine (short for ASCBS, ASCII scalar convert to binary scalar), ASCII convert binary, conversion of formats, noise generation is optional
- ACM: routine (short for AMCBM, ASCII matrix convert to binary matrix), ASCII convert matrix, conversion of formats
- ACS: routine (short for ASCAS, ASCII scalar convert to ASCII string), ASCII convert string, conversion of types

ACV: routine (short for AVCBV, ASCII vector convert to binary vector), ASCII convert vector, conversion of formats

ADD: main for ADD, adds two picture files to a sum file

ALLOC: routines tmalloc and talloc, tests whether space can be allocated and if so, it does, else programs stops

BCA: routine (short for BSCAS, short for binary scalar convert to ASCII scalar), binary convert ASCII, conversion of formats

BCV: routine (short for BSCBV, binary scalar convert to binary vector), binary convert vector, conversion of types, either according to thermometer or Gray coding

EIG: routine, calculates eigenvalues and eigenvectors

ENG: routine, calculates energy

GRPH: routines SetCursorPos and ReadCursorPos

IO: routines tfopen and tfopen, test whether files can be opened and if so, it does, else program stops

ISO: routine, calculates isoclines, optionally it also creates a memory dump or invokes PRNT to generate a print file

LRN: routine, learns a file and returns a weight file

MACB: main for ACB

MACM: main for ACM

MACS: main for ACS

MACV: main for ACV

MBCA: main for BCA

MBCV: main for BCV

MCA: routine (short for MBCMA, matrix binary convert to matrix ASCII), matrix convert ASCII, conversion of formats

MEIG: main for EIG

MENG: main for ENG

MISO: main for ISO

MJOB: main for JOB, a batch file for LRN and RCL

MLRN: main for LRN

MMCA: main for MCA

MPGN: main for PGN

MRCL: main for RCL

MSCA: main for SCA

MVCA: main for VCA

MVCB: main for VCB

MVCW: main for VCW
MWCV: main for WCV
PGN: *routine, find prototypes*
PLT: *routine, plots a picture file*
PRNT: *routine, translates a plot to a print file*
R2D: *read parameters for 2D*
R3D: *read parameters for 3D*
RACB: *read parameters for ACB*
RACV: *read parameters for ACV*
RBSB: *read general BSB parameters*
RCA: *main for RCA, (short for RBSCAS, radar binary scalar convert to ASCII scalar), radar convert ASCII, conversion of the format of radar file (generated by LOCK-ON simulator) to ASCII scalar format*
RCL: *routine, recalls a file given a weigh file*
RD: *read common parameters for 2D, 3D and ISO*
REIG: *read parameters for EIG*
RENG: *read parameters for ENG*
RFCP: *read parameter RFCP (fraction connection parameter, 1 is fully connected, 0 is not connected)*
RISO: *read parameters for ISO*
RJOB: *read parameters for JOB*
RNRF: *read parameter NRF*
RSOU: *read parameter source (whether ASCII or scalar, both for RCL and LRN)*
RUTL: *routine search, a read utility, searches in parameter file for a keyword, stops immediately after keyword if found, else program stops*
RVCW: *read parameter for VCW*
RWCV: *read parameter for WCV*
SCA: *routine (short for ASCAS, ASCII string convert to ASCII scalar), string convert to ASCII, conversion of types*
TEST: *main for TEST*
UTIL: *routines normalize, cmwv, rndcon, limit*
VCA: *routine (short for VBCVA, vector binary convert to vector ASCII), vector convert ASCII, conversion of formats*
VCB: *routine (short for VBCSB, vector binary convert to scalar binary), vector convert binary, conversion of types*

- VCW: routine (short for VBCWB, vector binary convert to weight binary), vector convert weight, generates the weight matrix
- WCV: routine (short for WBCVB, weight binary convert to vector binary), weight convert vector, given a weight matrix, it generates the BSB recall vector

Appendix B

Page
B.5

2 Structure of conversion executables

The mutual connection between the executables is given in figure B.1. In this figure datafiles are surrounded by squares and next to each line the name of an executable is given.

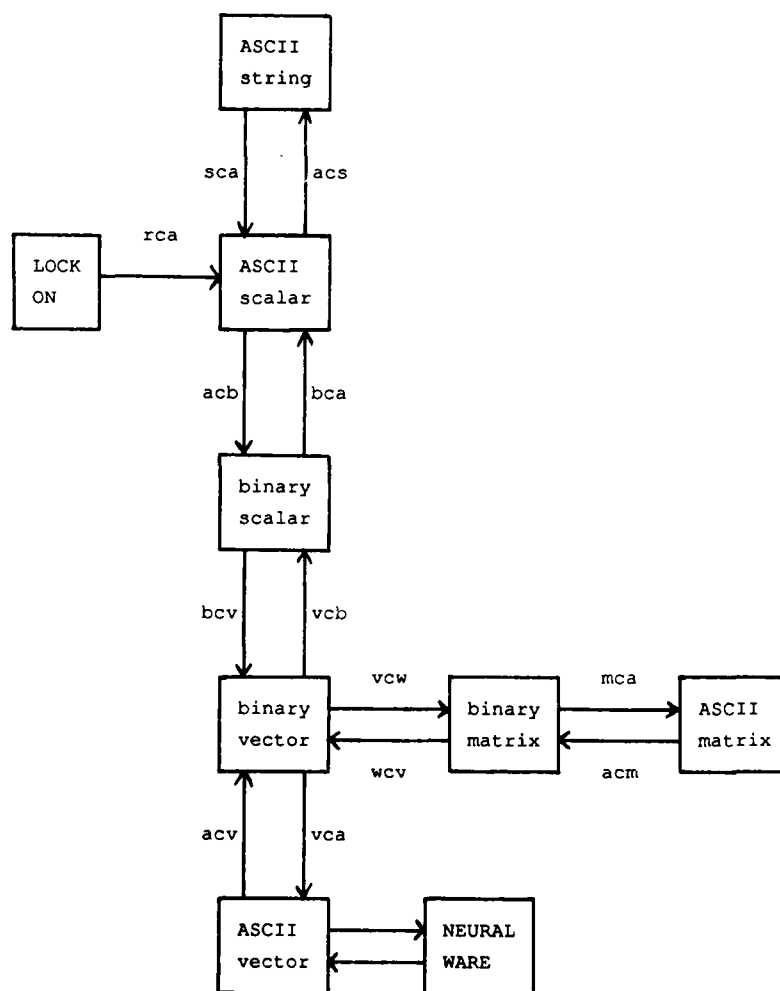


Figure B.1 Connections between the conversion programs.

The lrn program invokes the following programs, depending on the value of LrnSource (BSB.PAR). When LrnSource = string: sca, acb, bcv, vcw, when LrnSource = scalar: acb, bcv,vcw, when LrnSource = vector: acv, vcw. So lrn transforms an input file with a certain type to a binary matrix or weight file. The rcl program invokes the following programs, depending on the value of RclSource (BSB.PAR). When RclSource = string: sca, acb, bcv, wcv, vcb, bca, acs, when RclSource = scalar: acb, bcv, wcv, vcb, bca, when RclSource = vector acv, wcv, vca. So rcl transforms an input file with a certain type via a weight matrix to an output file with the same type as the input file.

UNCLASSIFIED

REPORT DOCUMENTATION PAGE

(MOD-NL)

1. DEFENSE REPORT NUMBER (MOD-NL) TD90-2780	2. RECIPIENT'S ACCESSION NUMBER	3. PERFORMING ORGANIZATION REPORT FEL-90-8023
4. PROJECT/TASK/WORK UNIT NO 20479	5. CONTRACT NUMBER	6. REPORT DATE AUGUST 1990
7. NUMBER OF PAGES 45 (incl. titlepage, appendices and RDP, excl. distr. list)	8. NUMBER OF REFERENCES 15	9. TYPE OF REPORT AND DATES FINAL REPORT
10. TITLE AND SUBTITLE BSB RADAR PULSE CLASSIFICATION		
11. AUTHOR(S) P.P. MEILER, A.M. VAN WEZENBEEK		
12. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) TNO PHYSICS AND ELECTRONICS LABORATORY P.O. BOX 96864 2509 JG THE HAGUE THE NETHERLANDS		
13. SPONSORING/MONITORING AGENCY NAME(S)		
14. SUPPLEMENTARY NOTES		
15. ABSTRACT THE BRAIN-STATE-IN-A-BOX (BSB) NEURAL NETWORK IS AN AUTO-ASSOCIATIVE NETWORK. IT IS ABLE TO ASSOCIATE VECTORS WITH OTHER VECTORS. AUTO-ASSOCIATION RESULTS IN A SYSTEM THAT HAS BEEN LEARNED TO RESPOND STRONGLY TO EACH INPUT VECTOR OF A CERTAIN SET OF INPUT VECTORS BY RETURNING THIS SAME VECTOR MULTIPLIED BY ITS RECOLLECTION STRENGTH, WHERE VECTORS ARE BOUNDED BY THE BOX. THIS GIVES THE POSSIBILITY FOR CONVERGENCE TO ONE VECTOR OF THE LEARNED SET OF VECTORS IF THE ACTUAL INPUT VECTOR IS NOT KNOWN TO THE SYSTEM. WHEN RADAR SIGNALS ARE TRANSLATED INTO VECTORS, THE BSB NETWORK CAN BE USED FOR RADAR PULSE CLASSIFICATION. THE WHOLE PROCESS OF CLASSIFICATION TOGETHER WITH THE RELATION WITH OTHER NON-NEURAL ALGORITHMS IS SHOWN.		
16. DESCRIPTORS NEURAL NETWORKS PARALLEL PROCESSING PATTERN CODING PATTERN RECOGNITION PULSE RADAR	IDENTIFIERS AUTO-ASSOCIATION BRAIN-STATE-IN-A-BOX (BSB) NEURAL NETWORK CLUSTERING GRADIENT DESCENT RADAR PULSE CLASSIFICATION	
17a. SECURITY CLASSIFICATION (OF REPORT) UNCLASSIFIED	17b. SECURITY CLASSIFICATION (OF PAGE) UNCLASSIFIED	17c. SECURITY CLASSIFICATION (OF ABSTRACT) UNCLASSIFIED
18. DISTRIBUTION/AVAILABILITY STATEMENT UNLIMITED AVAILABILITY	17d. SECURITY CLASSIFICATION (OF TITLES) UNCLASSIFIED	

UNCLASSIFIED